Algorithmentechnik — Übung 3

http://i11www.ira.uka.de/teaching/WS_0506/algotech

Steffen Mecke (mecke@ira.uka.de)

WS 05/06

Übersicht

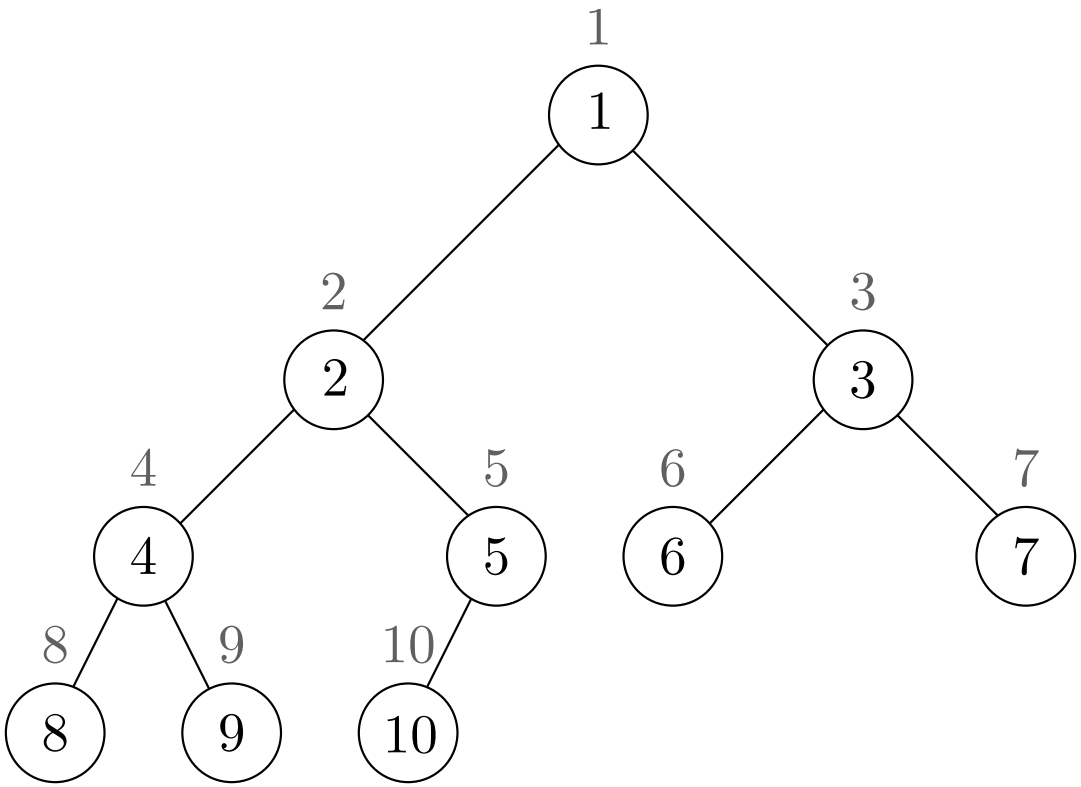
Heapsort

Algorithmus 1 : Heapsort (A)

Eingabe : Array A der Länge n
Ausgabe : Aufsteigend sortiertes Array A

```
1 MAKEHEAP(A);
2 für i = n, ..., 2 tue
3 |   Vertausche A[1] und A[i];
4 |   HEAP-Größe(A) ← HEAP-Größe(A) − 1;
5 |   HEAPIFY (A, 1);
```

Beispiel 1



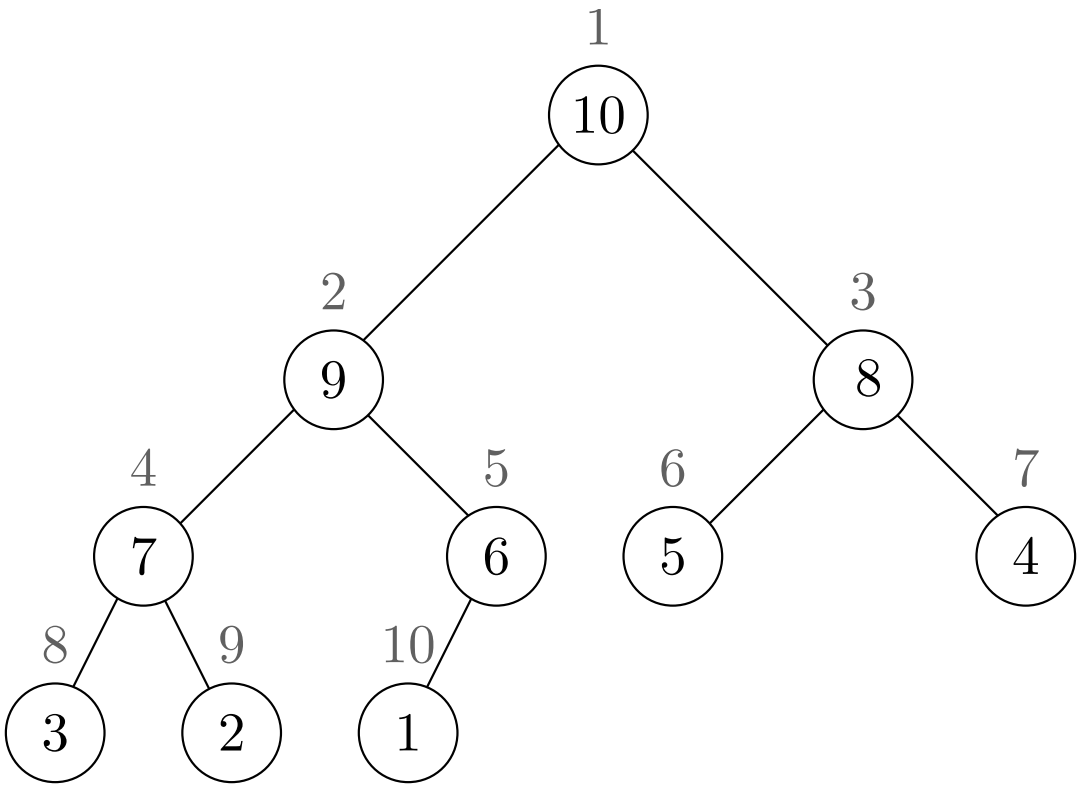
$A =$

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

Aufsteigend sortiertes Array

- Zu Anfang sind die $n/2$ größten Elemente in Blättern.
- Durch MAKEHEAP werden mindestens die Hälfte ($n/4$) von ihnen nach oben getauscht.
- Im Laufe der ersten $n/2$ Schleifendurchläufe werden diese Knoten nur nach oben steigen, denn
- in jedem Schritt wird jeweils ein Blatt nach oben getauscht und HEAPIFY aufgerufen.
- Die $n/2$ größten Knoten werden dabei nach und nach aus dem HEAP entfernt, müssen also nach $A[1]$ aufsteigen.
- Insgesamt steigen also $n/4$ Knoten um mindestens $\log n - 1$ Level auf.
- Dafür sind mindestens $n/4 \cdot (\log n - 1) = \Omega(n \log n)$ Vergleiche nötig.

Beispiel 2



$A =$

1	2	3	4	5	6	7	8	9	10
10	9	8	7	6	5	4	3	2	1

Absteigend sortiertes Array

- Absteigend sortiertes Array ist schon ein HEAP.
- Die $n/2$ kleinsten Elemente sind in Blättern.
- Sie werden in den ersten $n/2$ Schleifendurchläufen nach oben getauscht und nicht aus dem HEAP entfernt.
- Wie weit sinken sie nach unten?

Wie weit sinken kleine Elemente?

- Betrachte die obere Hälfte des HEAP, also alle Knoten mit Tiefe höchstens $(\log n)/2$.
- Dies sind $2^{\lfloor \lfloor \log n \rfloor /2 \rfloor} - 1 \leq (2^{\lfloor \log n \rfloor})^{1/2} \leq \sqrt{n}$ Knoten.
- Aus Platzmangel an der Spitze sinken mindestens $n/2 - \sqrt{n}$ kleine Elemente *tiefer als* $(\log n)/2$.
- Aufwand dafür mindestens $(n/2 - \sqrt{n}) \cdot (\log n)/2 = \Omega(n \log n)$.

Best-case-Laufzeit

- Wir wissen: Kein Sortieralgorithmus, der auf Vergleichen basiert, ist schneller als $\Omega(n \log n)$ im Worst-case!
- Ist der Best-case besser?
- Fall 1: Nach MAKEHEAP sind mindestens $n/4$ „große Werte“ (aus der größeren Hälfte) oberhalb der Blätter.
  - Aufwand durch HEAPIFY: $(n/4 - \sqrt{n}) \cdot (\log n)/2 = \Omega(n \log n)$.
- Fall 2: Mindestens $n/4$ „große Werte“ sind Blätter.
  - Alle Werte oberhalb eines Blattes sind größer als der Wert des Blattes.
  - In der Schicht oberhalb der $n/4$ „großen Blätter“ sind mindestens $n/8$ große Werte.
  - Aufwand durch HEAPIFY: $(n/8 - \sqrt{n}) \cdot (\log n)/2 = \Omega(n \log n)$.

Increasekey

Algorithmus 2 : Heap-Increasekey(A, i, k)

Eingabe : Array A als HEAP, Index i, Wert k
Ausgabe : Array A (Wert $A[i]$ aktualisiert) als HEAP
$A[i] \leftarrow \max(k, A[i])$;
SIFT-UP (A, i);

Mergelists

- Eingabe: k (absteigend) sortierte Listen.
- Ausgabe: Eine sortierte Liste aller Elemente.
  1. Bilde HEAP aus den k ersten Elementen der k Listen.
  2. Wiederhole
  3. Entferne Maximum aus dem HEAP
  4. „Lade“ aus der dazugehörigen Liste das nächstgrößte Element nach (falls vorhanden).
  5. HEAPIFY
  6. bis alle Listen leer sind.
- Die extrahierten Elemente sind absteigend sortiert.
- Aufwand: n-mal HEAPIFY auf einem HEAP der Größe k, insgesamt $\Theta(n \log k)$.

Bäume

Definition

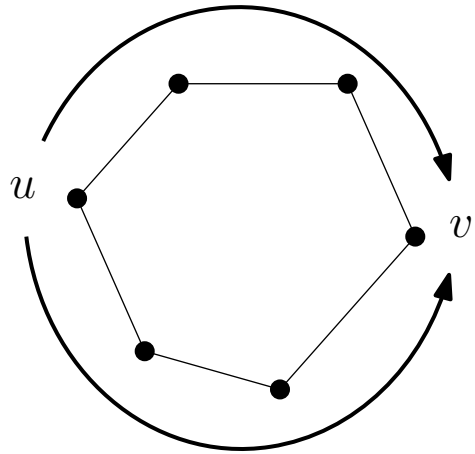
Ein *Baum* ist ein Graph, in dem zwischen je zwei Knoten genau ein Pfad existiert.

Definition (Äquivalente Definitionen)

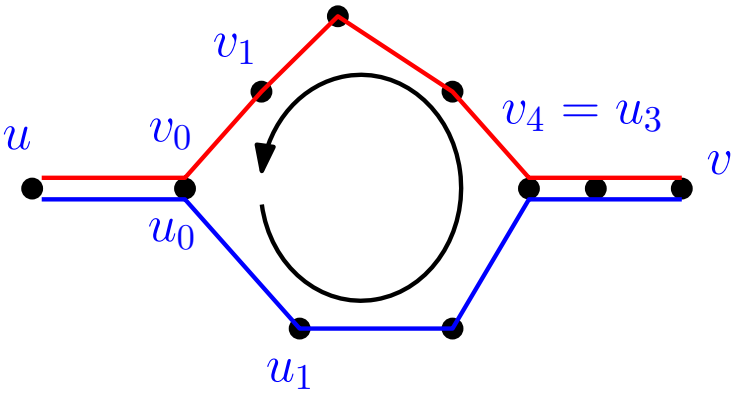
- Ein Baum ist ein maximal kreisfreier Graph.
- Ein Baum ist ein minimal zusammenhängender Graph.
- Ein Baum ist ein zusammenhängender Graph ohne Kreise.

Zwei Tatsachen zu Kreisen und Pfaden

Ein Kreis induziert zwei kantendisjunkte Pfade:



Zwei verschiedene Pfade induzieren einen Kreis:



Sei $G = (V, E)$ ein Graph. Dann sind äquivalent:

(a) Zwischen je zwei Knoten existiert genau ein Pfad.

(b) G ist ein maximal kreisfreier Graph.

(a) $\Rightarrow$ (b): Zu zeigen:

- G enthält keinen Kreis und
- durch Hinzunahme einer Kante entsteht ein Kreis.

Beweis.

- Würde ein Kreis existieren, dann gäbe es zwei Pfade.
- Durch Hinzunahme einer Kante (u, v) entsteht ein neuer Pfad von u nach v. Zusammen mit dem bereits in G existierenden Pfad entsteht so ein Kreis. □

Sei $G = (V, E)$ ein Graph. Dann sind äquivalent:

(a) Zwischen je zwei Knoten existiert genau ein Pfad.

(b) G ist ein maximal kreisfreier Graph.

(b) $\Rightarrow$ (a): Zu zeigen:

- Zwischen je zwei Knoten existiert ein Pfad.
- Zwischen keinen zwei Knoten existieren zwei verschiedene Pfade.

Beweis.

- Seien u und v zwei beliebige Knoten. Falls die Kante (u, v) existiert, ist dies ein Pfad. Sonst entsteht durch Hinzunahme von (u, v) ein Kreis. Also gibt es zwei Pfade. Nur einer davon geht über (u, v). Der andere existiert schon in G.
- Würden zwei Pfade zwischen zwei Knoten existieren, so hätten wir einen Kreis. □

Bemerkung

Ein Baum mit n Knoten hat $n-1$ Kanten.

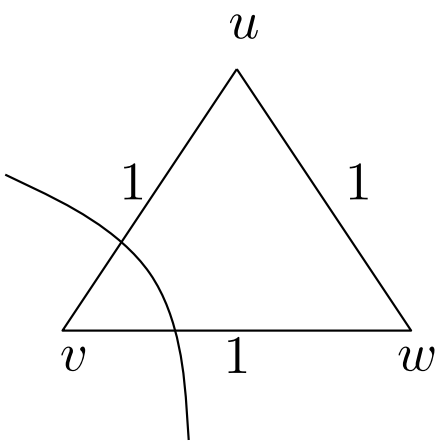
Beweis.

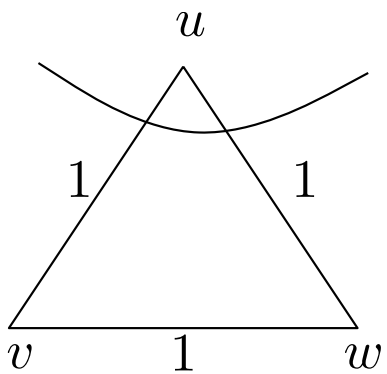
- Induktion über $n = |V|$.
- $n = 1$: klar, $n = 2$: Es gibt genau eine Kante.
- $n > 2$: Betrachte einen längsten Pfad $(v_0, v_1, v_2, \ldots, v_{l-1}, v_l)$. v_0 ist ein Blatt (Knoten von Grad 1). [Sonst gäbe es einen längeren Pfad oder einen Kreis, d.h zwei Pfade von v_0 zu einem v_i.]
- $G - \{v_0, \{v_0, v_1\}\}$ hat nach Induktionsvoraussetzung $n-2$ Kanten.
- Also: G hat $n-1$ Kanten. □

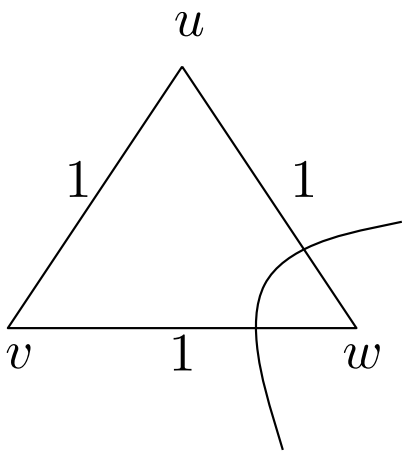
Leichte Kanten

Eine Kante e heiße *leicht*, wenn es einen Schnitt gibt, so dass e unter allen Kanten, die diesen Schnitt kreuzen, minimales Gewicht hat. Geben Sie einen gewichteten Graphen an, so dass die Menge der leichten Kanten keinen minimal aufspannenden Baum induziert.

Alle Kanten in folgendem Graphen sind leicht. Alle leichten Kanten induzieren jedoch ein Dreieck.



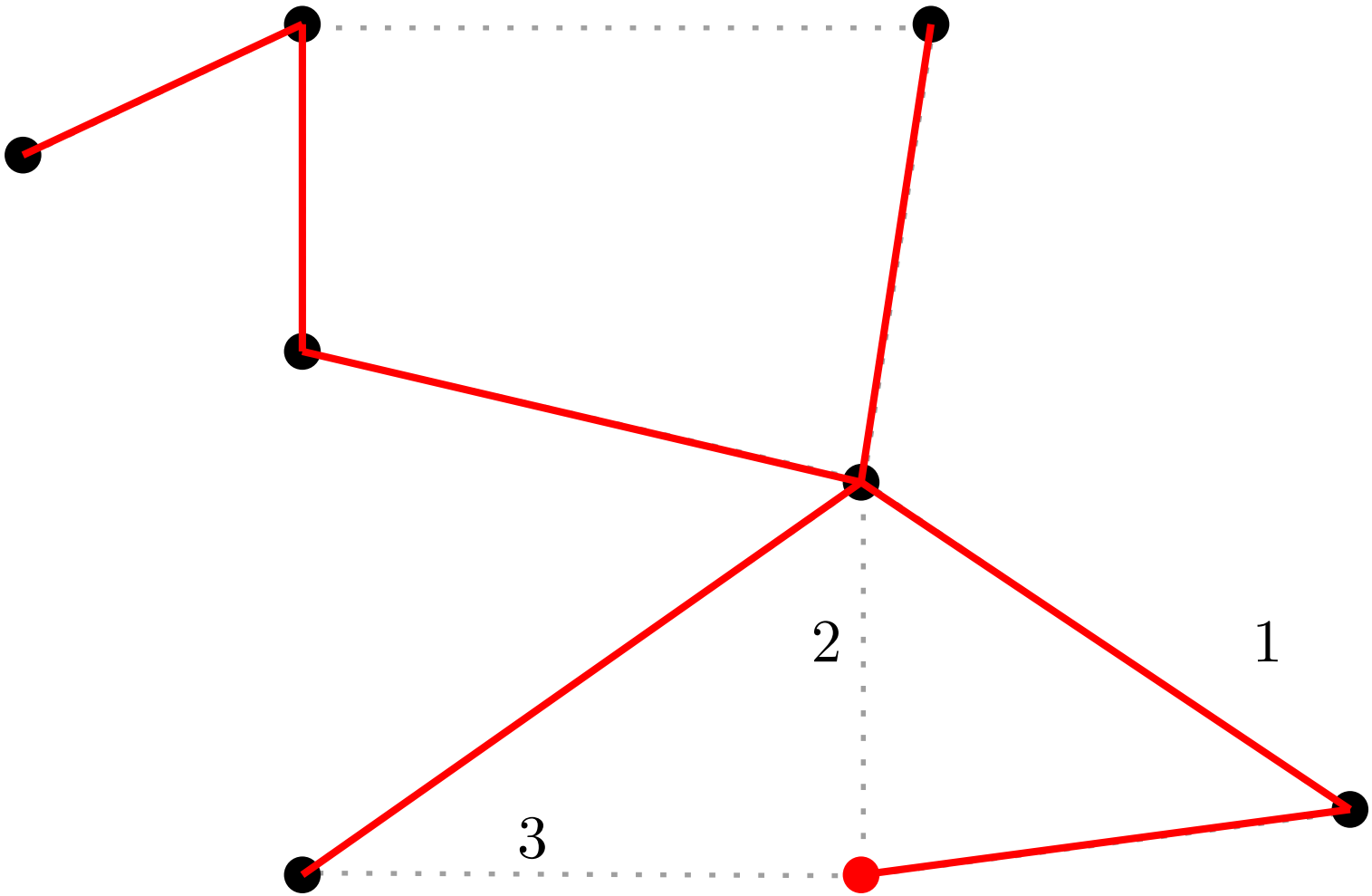




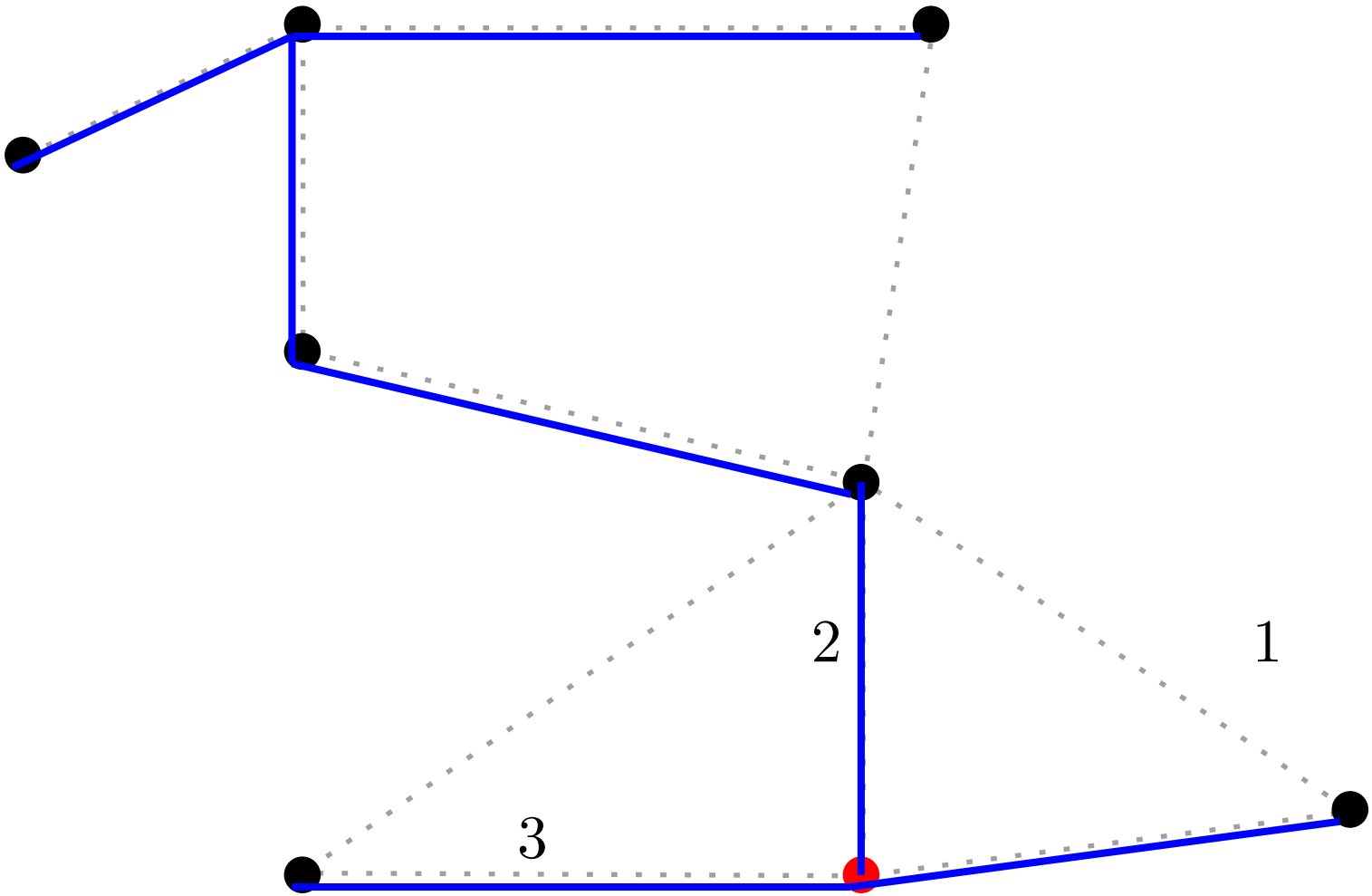
Eindeutiger MST

Für jeden Schnitt sei die den Schnitt kreuzende Kante minimalen Gewichts eindeutig. Dann hat G einen eindeutigen MST.

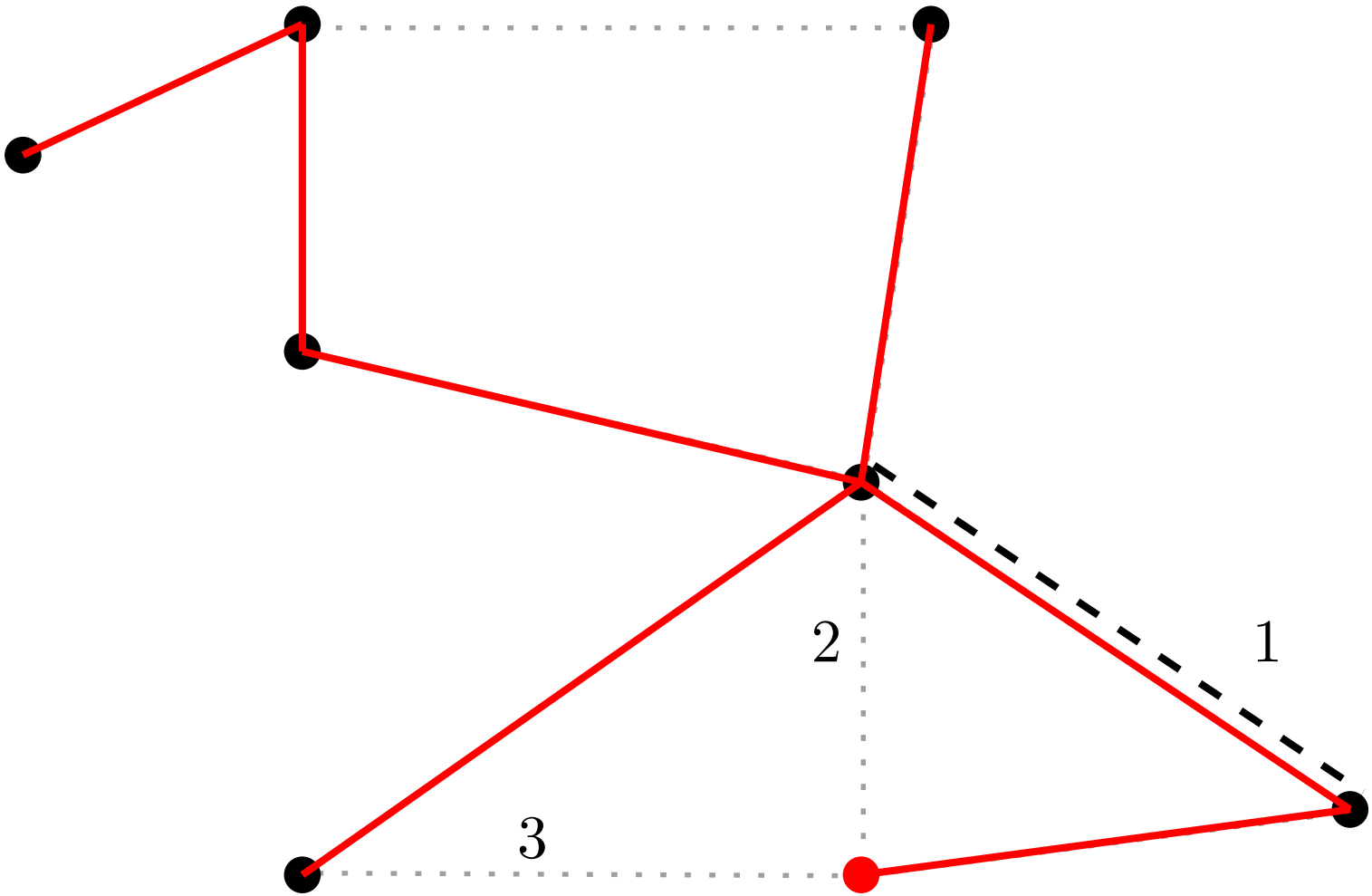
Beweis



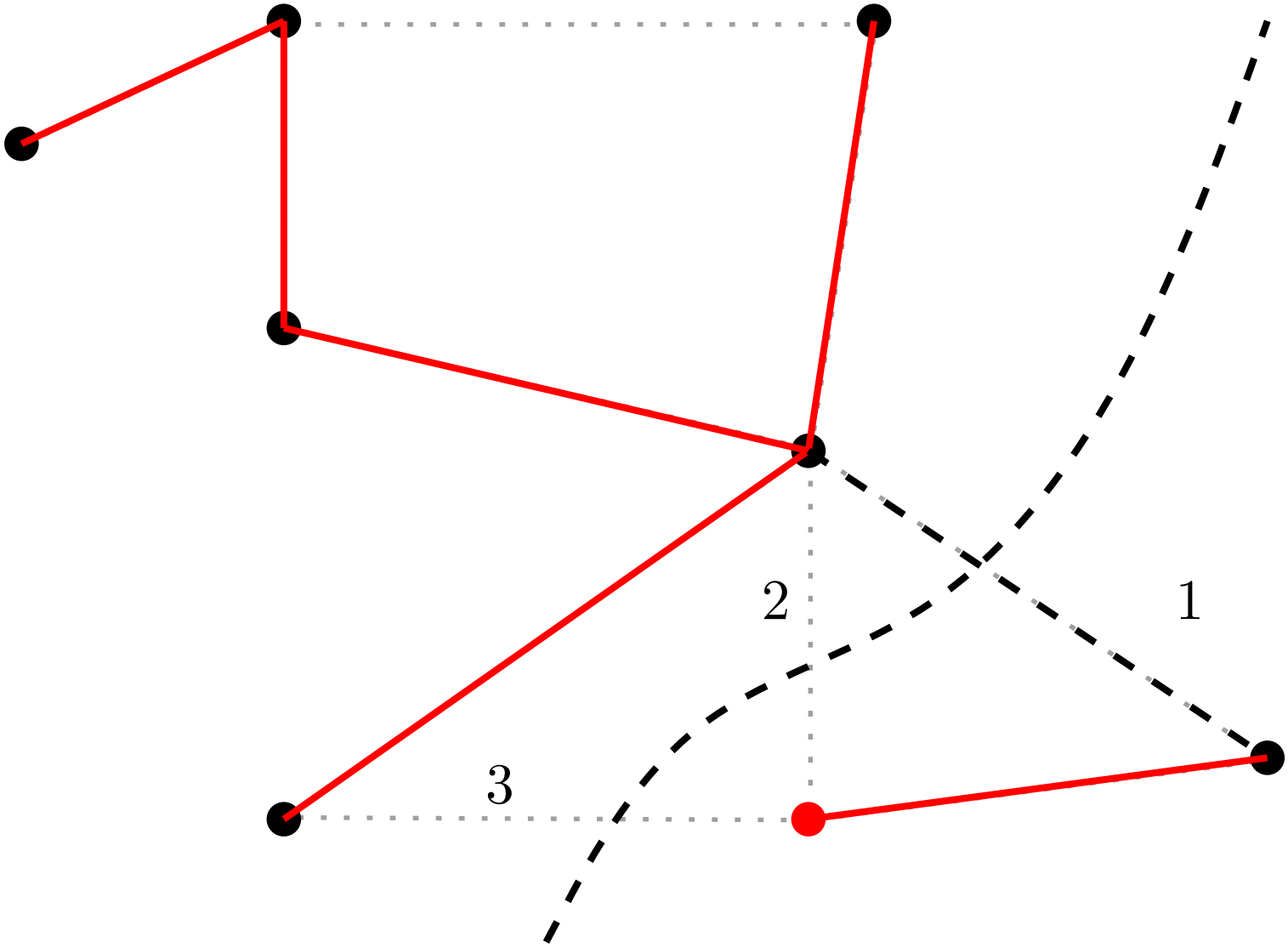
Beweis



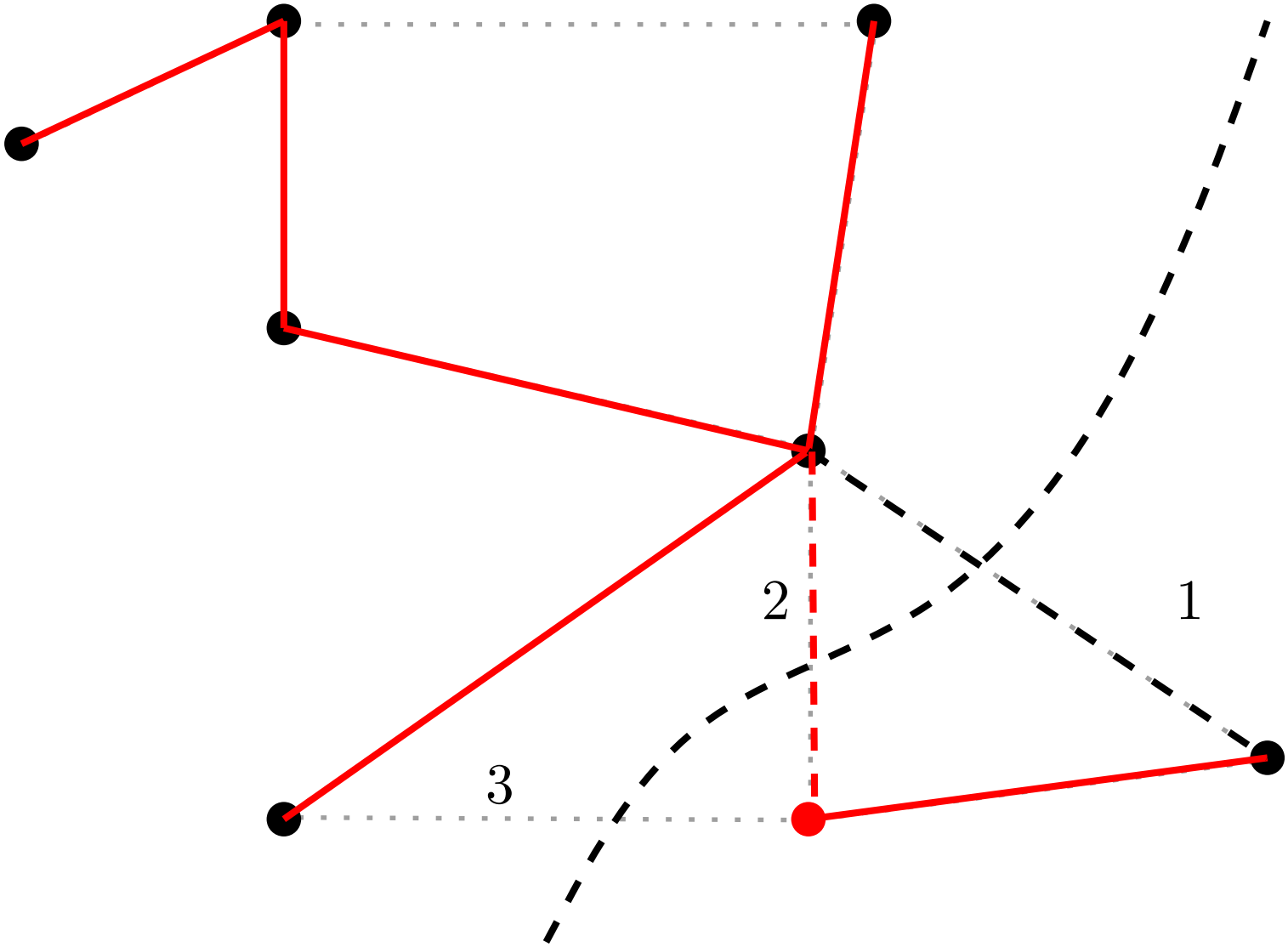
Beweis



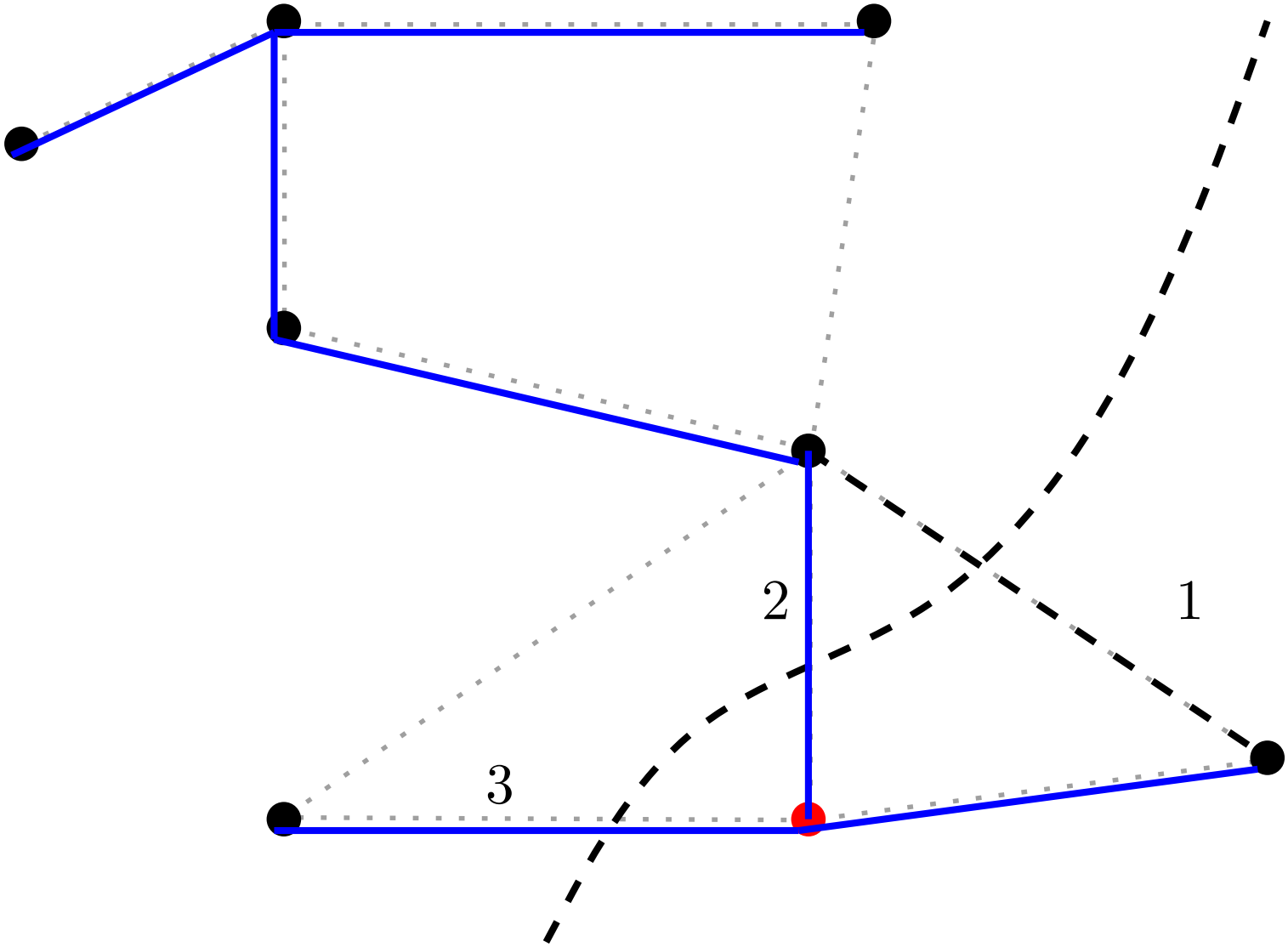
Beweis



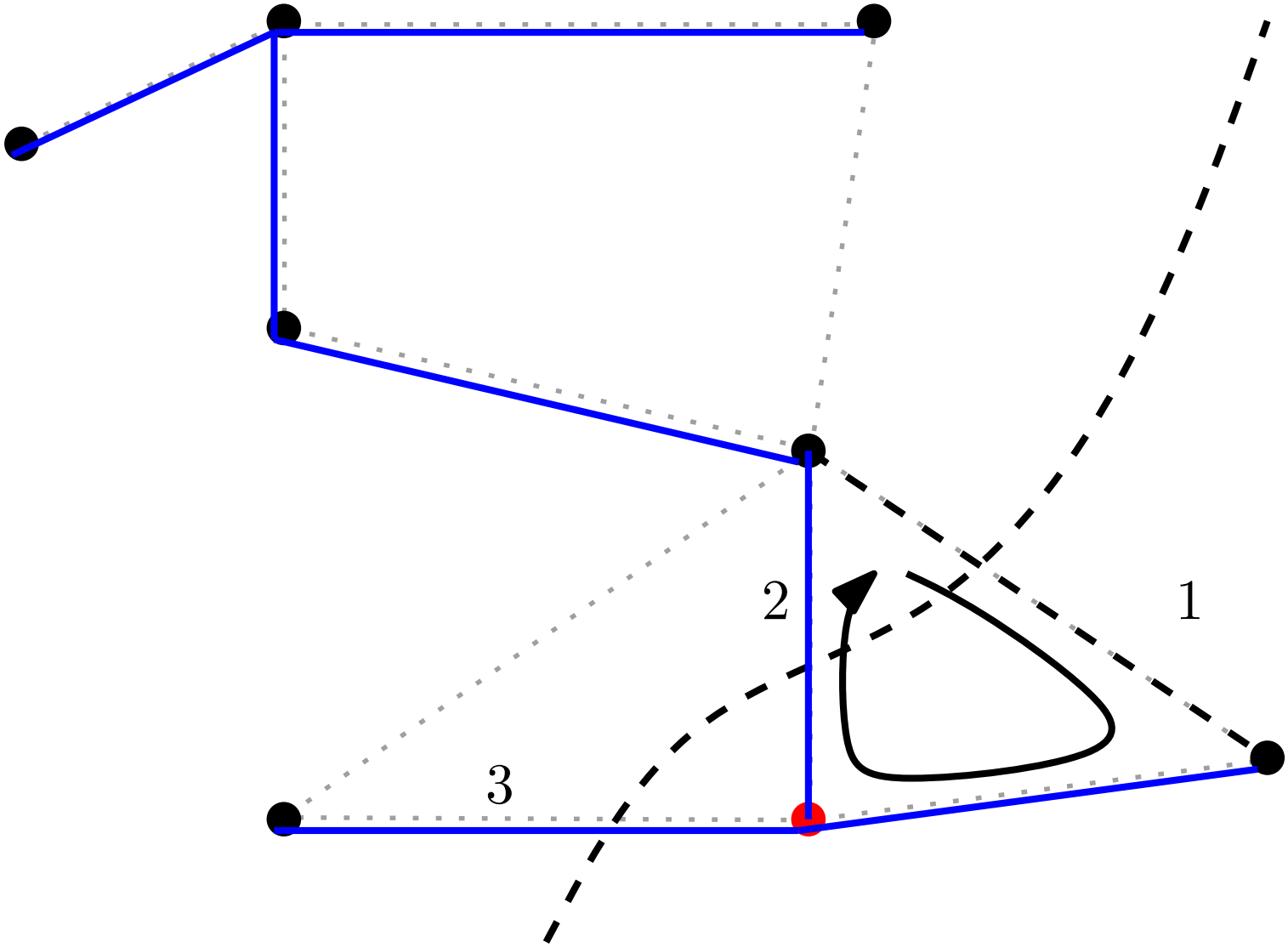
Beweis



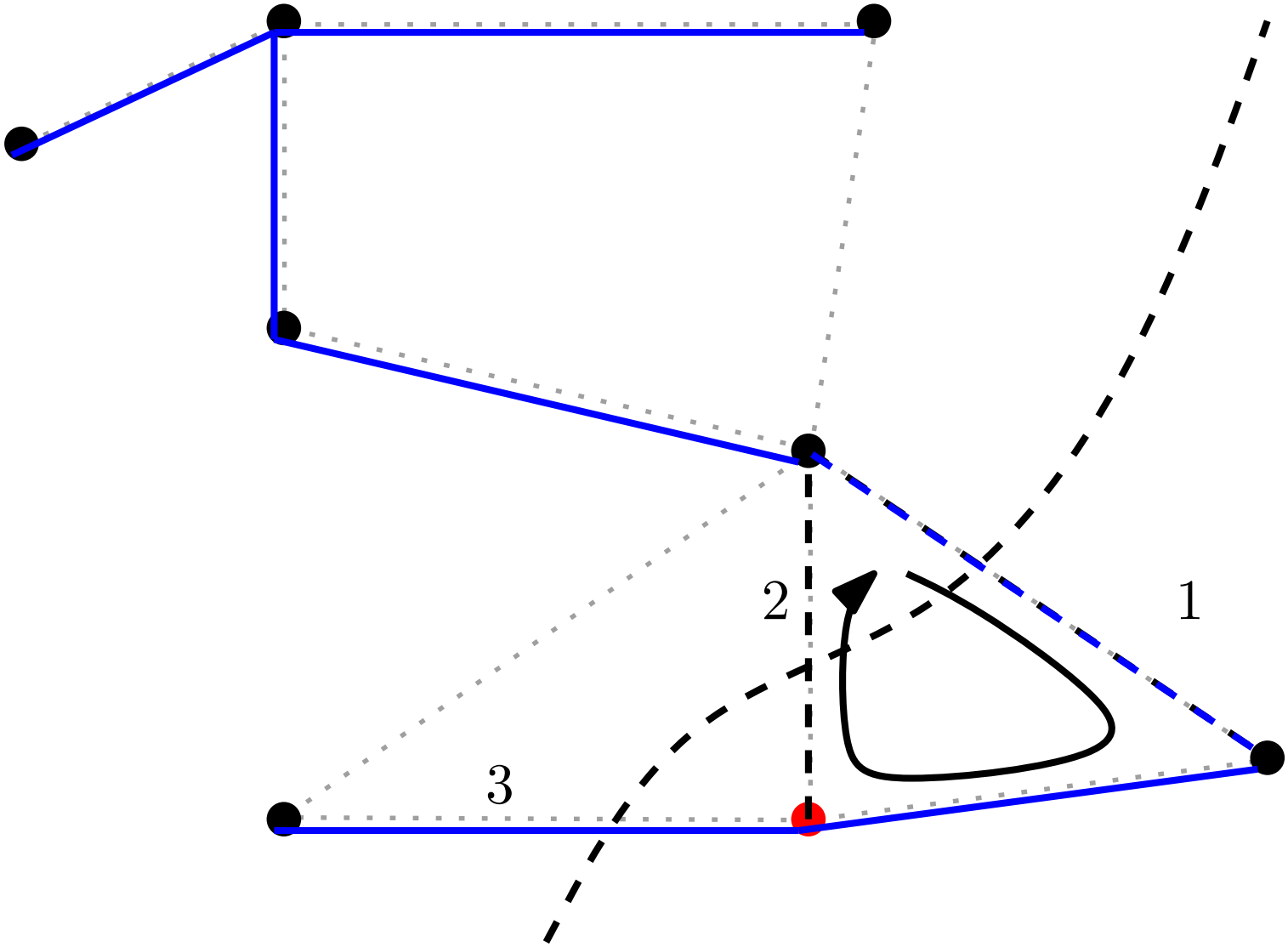
Beweis



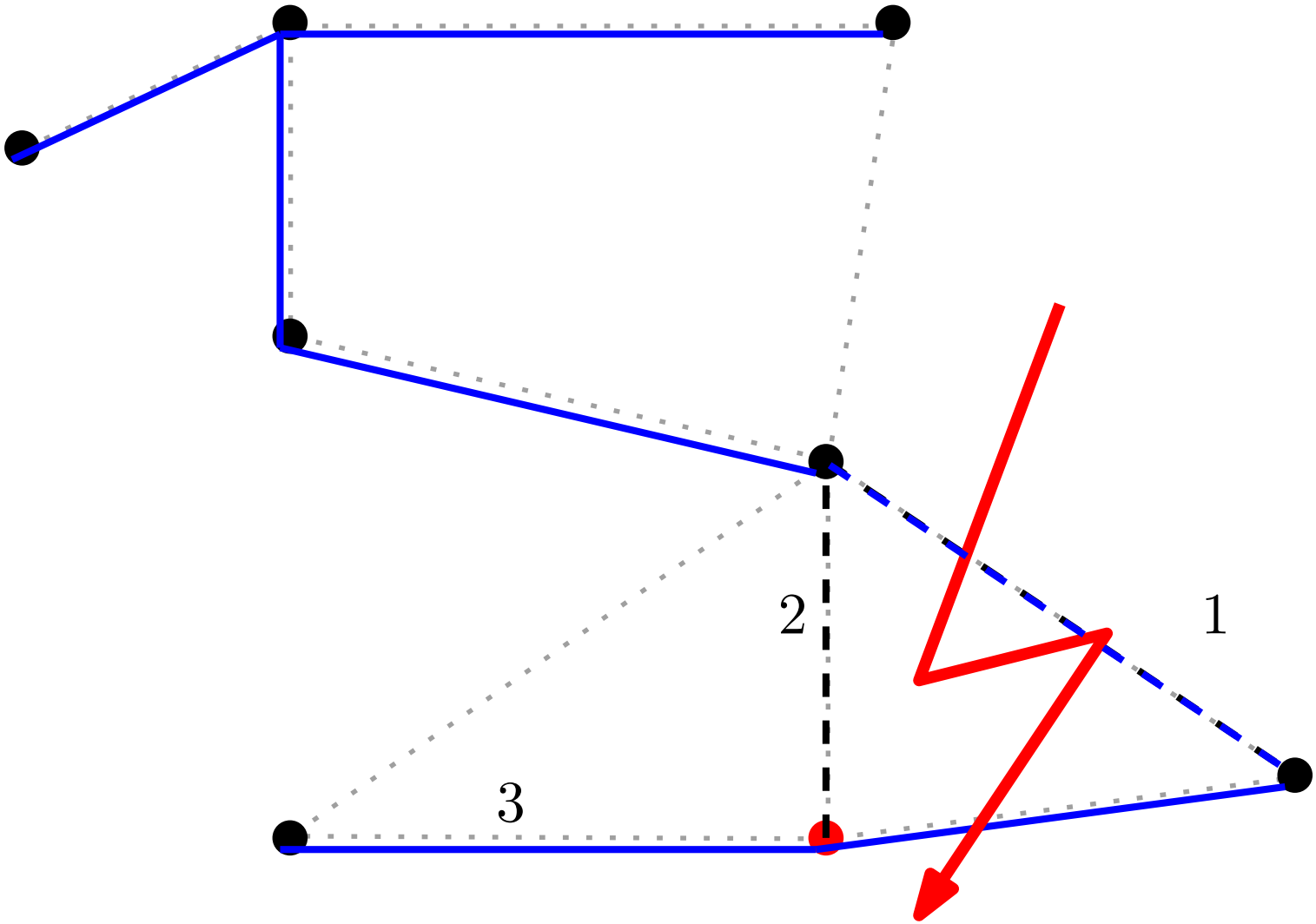
Beweis



Beweis



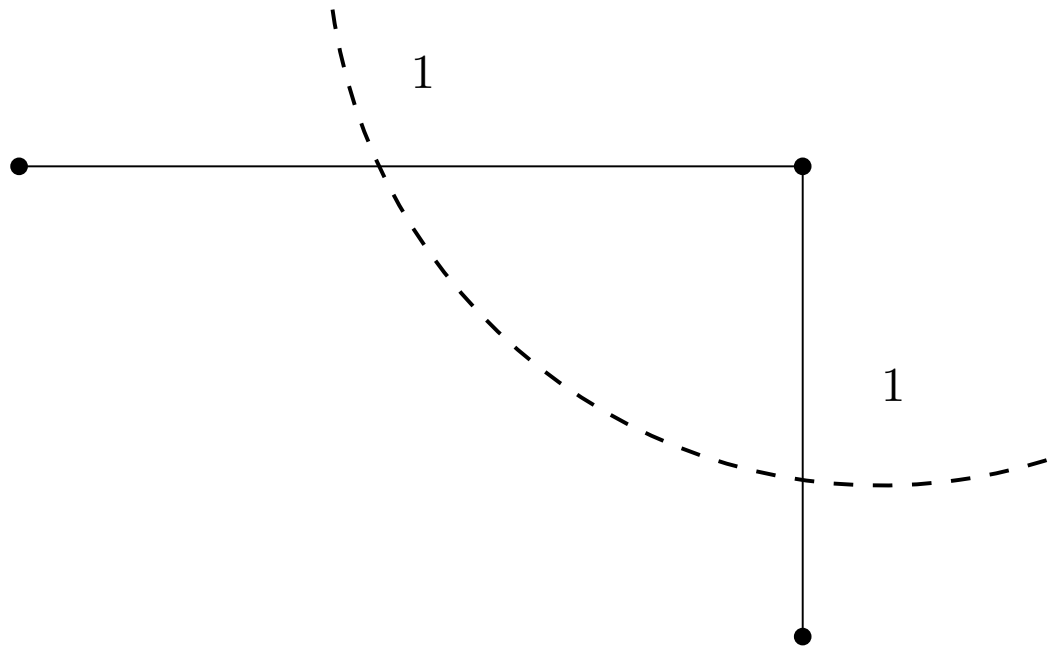
Beweis



Umkehrung

G habe einen eindeutigen MST. Dann ist für jeden Schnitt die den Schnitt kreuzende Kante minimalen Gewichts eindeutig??

Nein!



In G haben seien alle Kantengewichte verschieden. Dann hat G einen eindeutigen MST.

Folgt aus vorigem: Alle Kanten, die einen Schnitt kreuzen haben verschiedenes Gewicht. Also gibt es ein eindeutiges Minimum. Also ist der MST eindeutig.

Umkehrung gilt nicht (selbes Gegenbeispiel).

Matroide

Matroide sind eine Verallgemeinerung des Konzepts der „linearen Unabhängigkeit“ in Vektorräumen.
Erinnerung: Für eine Grundmenge E ist $(E, \mathcal{U})$ mit $\mathcal{U} \subset \mathcal{P}(E)$ ein Matroid falls gilt:

1. „Die leere Menge ist linear unabhängig.“
$$\emptyset \in \mathcal{U}$$
2. „Teilmengen linear unabhängiger Mengen sind linear unabhängig.“
$$I \subset J,\ J \in \mathcal{U} \Rightarrow I \in \mathcal{U}$$
3. „Basisergänzungssatz: Eine kleine linear unabhängige Menge lässt sich mit Hilfe einer grösseren verlängern.“
$$I \in \mathcal{U},\ J \in \mathcal{U},\ |I| < |J| \Rightarrow \exists x \in J \setminus I : I \cup \{x\} \in \mathcal{U}$$

Aussagen über Matroide

- „Basisaustauschsatz“:

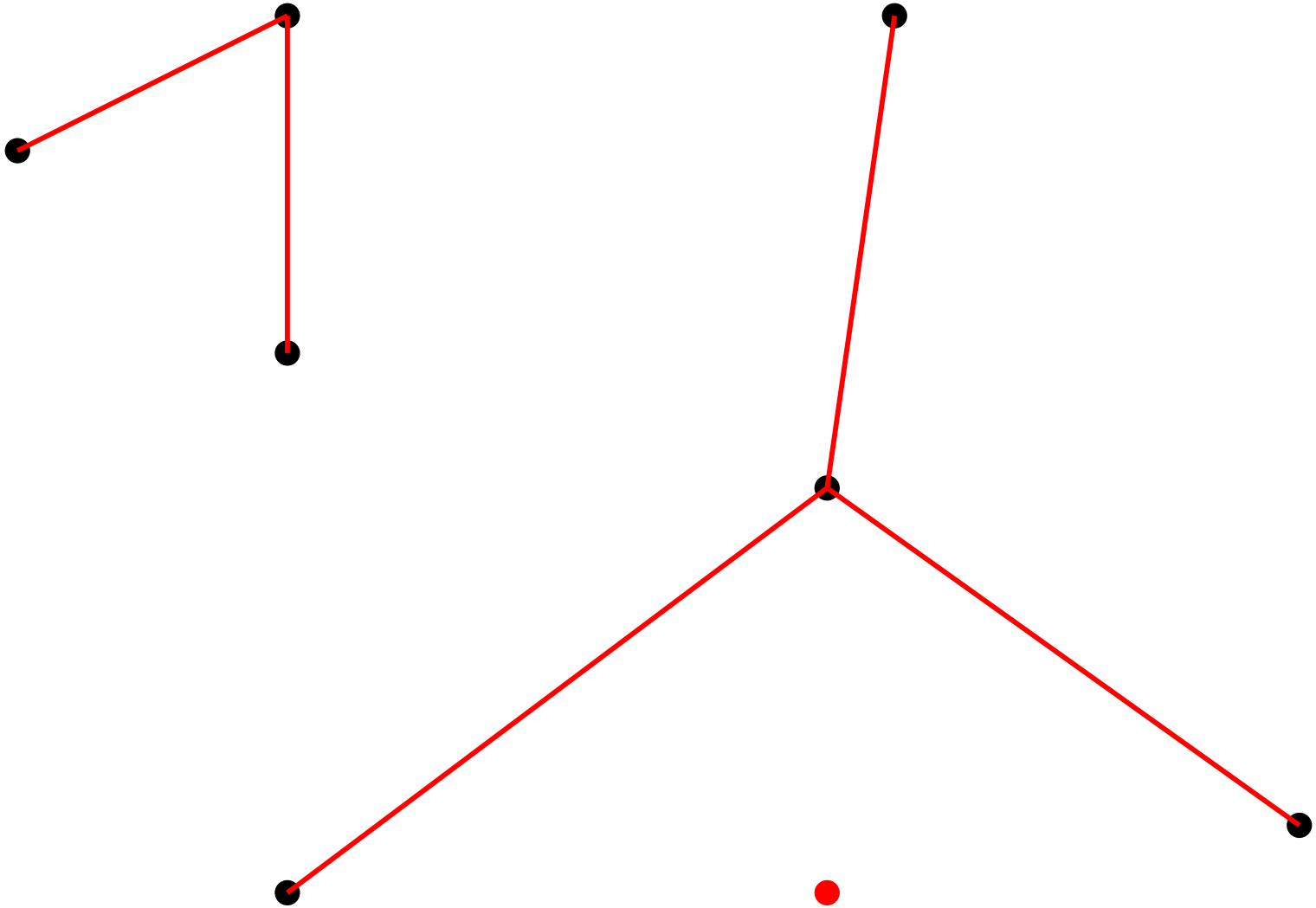
$$I \in \mathcal{U},\ J \in \mathcal{U},\ |I| = |J| \Rightarrow \exists x \in I\ \exists y \in J\colon I \setminus \{x\} \cup \{y\} \in \mathcal{U}$$

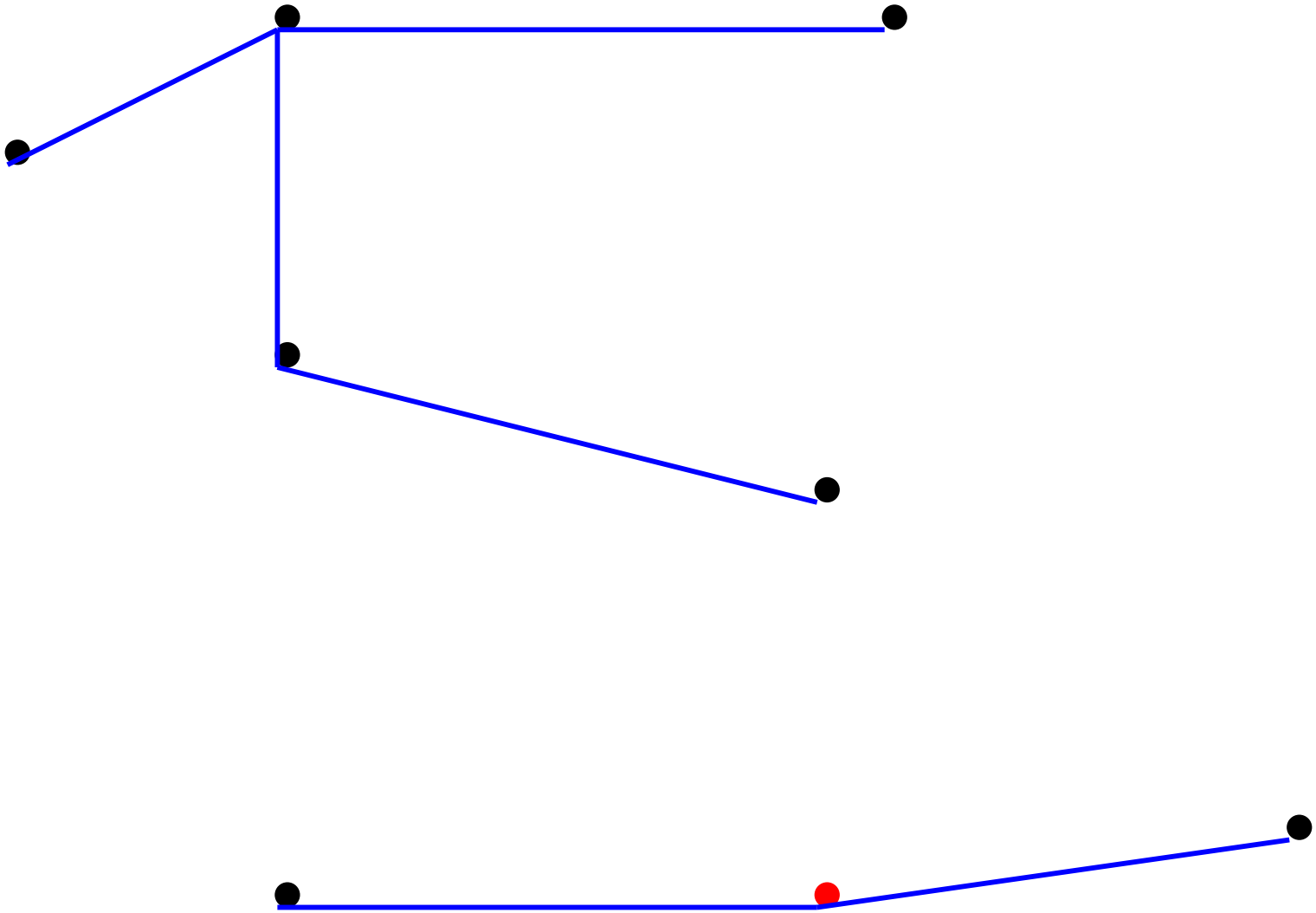
- „Alle Basen sind gleich lang.“
  Für zwei inklusionsmaximale unabhängige Mengen („Basen“) $I_1, I_2 \subseteq \mathcal{U}$ gilt $|I_1| = |I_2|$.

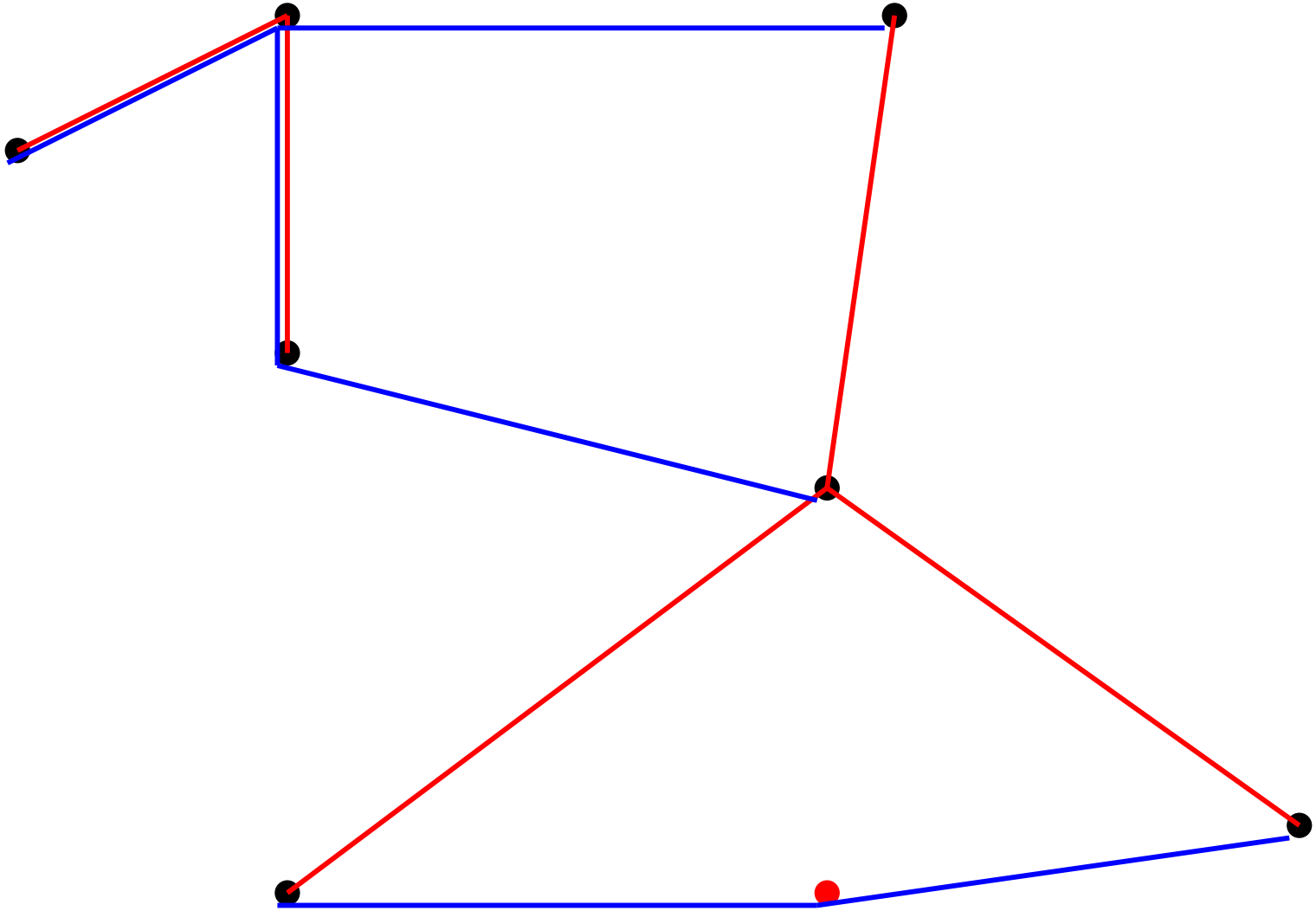
Problem 6

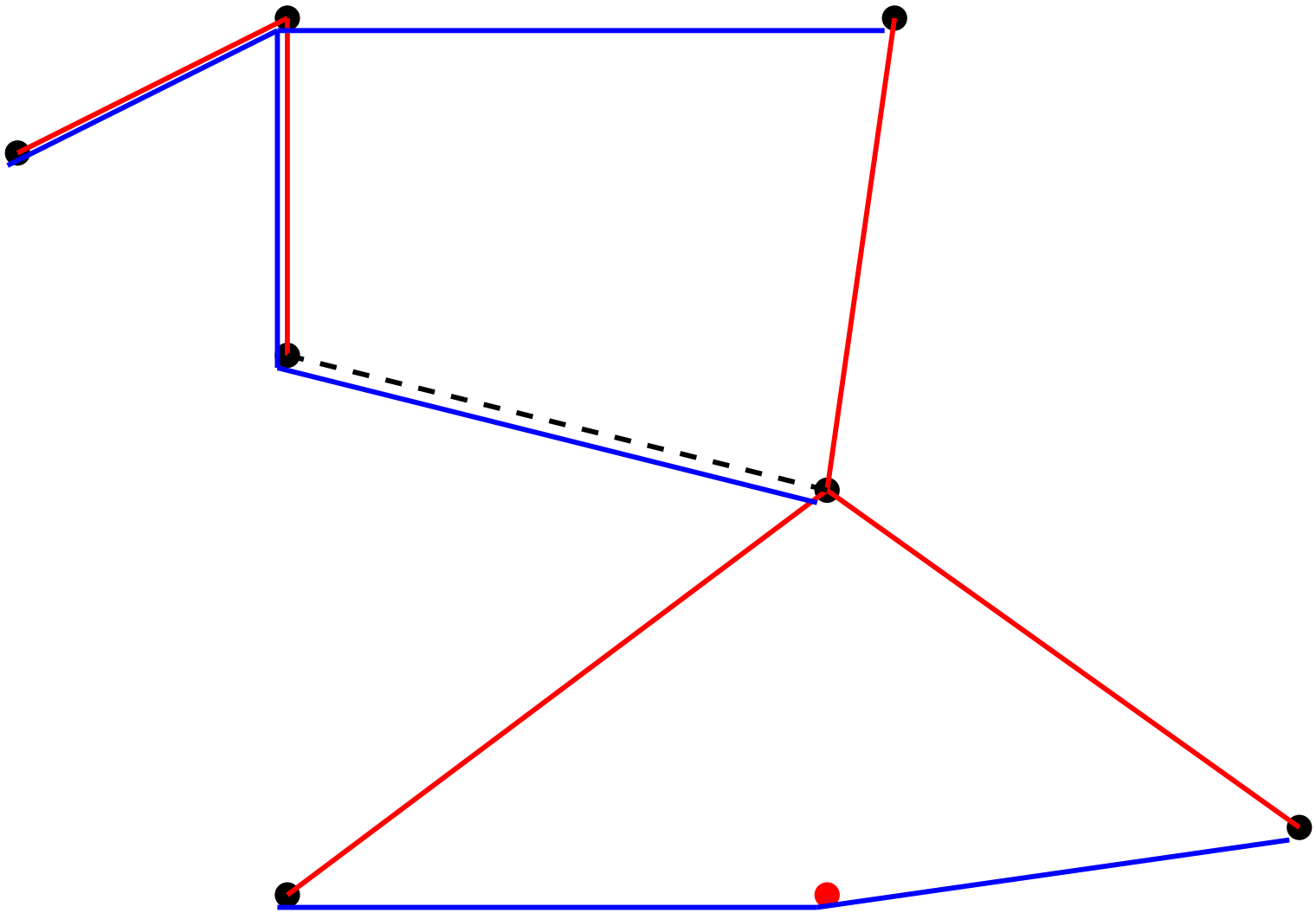
Sei $G = (V, E)$ ein Graph, $U := \{E_T \subset E : (V, E_T) \text{ ist ein Wald}\}$. [Ein Graph ist ein Wald, wenn zwischen je zwei Knoten *höchstens* ein Pfad existiert.]

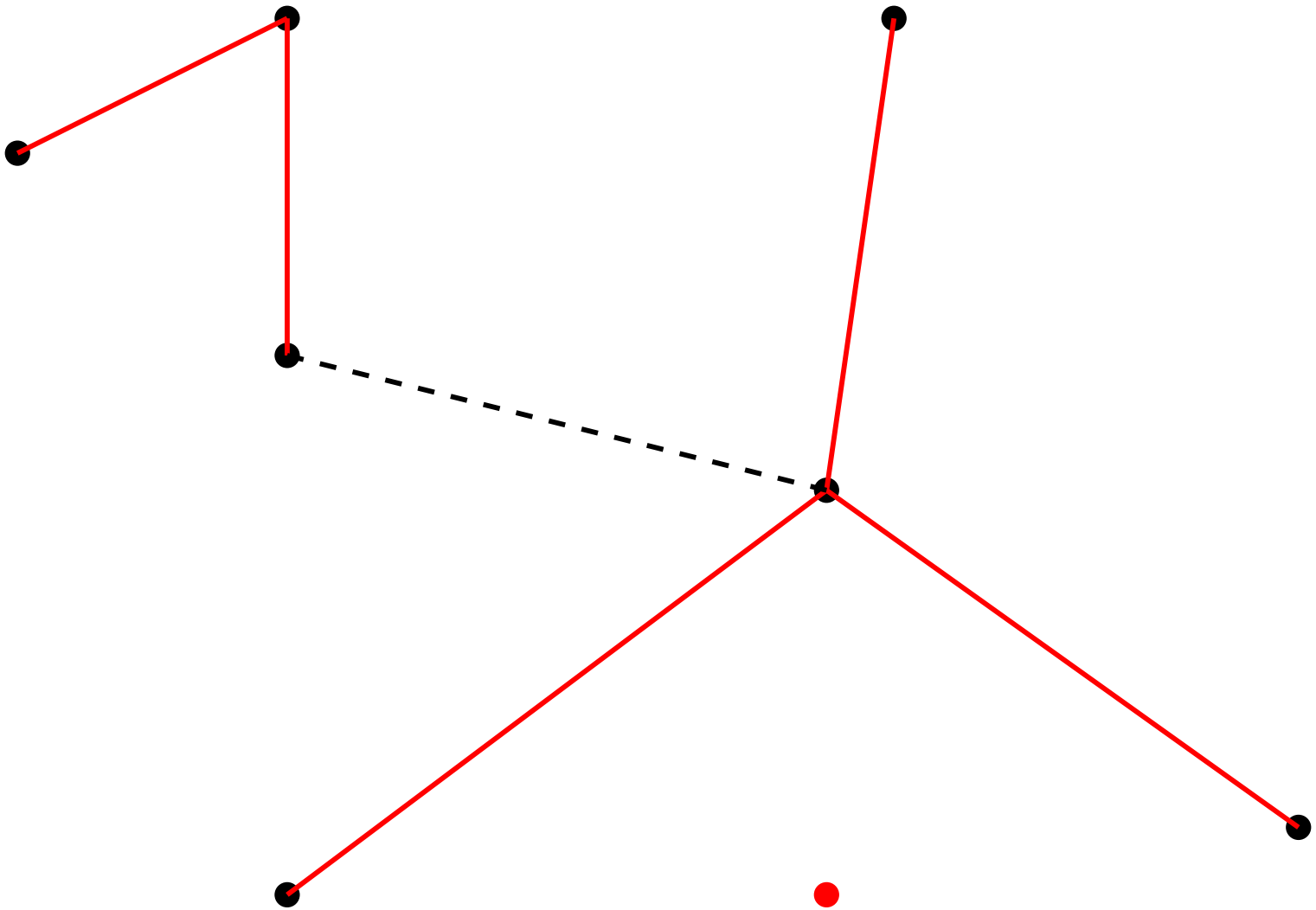
1. Die leere Kantenmenge induziert $|V|$ Bäume.
2. Sei $I \subset J,\ J \in \mathcal{U}$. Dann existiert in (V, I) immer noch höchstens ein Pfad zwischen zwei Knoten.
3. Seien $I \in \mathcal{U},\ J \in \mathcal{U},\ k := |I|,\ \ell := |J|$ und $k < \ell$.
   - I induziert $n - k$ Zusammenhangskomponenten (Bäume) in G (Beweis durch Induktion). J induziert $n - \ell$ Bäume.
   - $I \cup J$ induziert höchstens $n - \ell < n - k$ Komponenten.
   - Also gibt es mindestens eine Kante $e \in J$ die zwei Zusammenhangskomponenten von I verbindet.
   - Nehmen wir diese Kante zu I hinzu, erhalten wir wieder eine Menge von Bäumen, also gilt $I \cup \{e\} \in U$.











Einmaschinenscheduling

- Eine Reihe von Aufträgen $A_1, \ldots, \mathcal{A}_n$.
- *Eine* Ressource (Maschine).
- Jeder Auftrag dauert *eine* Zeiteinheit. Also dauern k Aufträge k Zeiteinheiten. Start bei Zeit 0.
- Jeder Auftrag hat eine Zeitlimit $D_i \geq 0$ bis zu dem er fertig sein muss. Zur Vereinfachung sei $D_1 \leq D_2 \leq \cdots \leq D_n$.
- Das Mengensystem aller Teilmengen von Aufträgen, die rechtzeitig bearbeitet werden können ist ein Matroid.

Beweis I

- Eine Teilmenge $\{A_{i_1}, \ldots, A_{i_k}\}$ von k Aufträgen ist zulässig genau dann, wenn

$$\forall j \leq k : D_{i_j} \geq j \qquad (*)$$

1. Eine leere Menge von Aufträgen kann stets rechtzeitig bearbeitet werden.
2. Eine Teilmenge einer zulässigen Menge von Aufträgen kann ebenfalls rechtzeitig bearbeitet werden.

Beweis II

3. Seien $\mathcal{I} := \{A_{i_1}, \ldots, A_{i_k}\}$ und $\mathcal{J} := \{A_{j_1}, \ldots, A_{j_\ell}\}$ zwei zulässige Mengen mit $k < \ell$.
   - Falls $A_{i_k} \neq A_{j_\ell}$ dann kann A_{j_ℓ} an I angehängt werden, da $|J| > |I|$ und J zulässig.
   - Falls $A_{i_k} = A_{j_\ell}$ dann laufe in I und J solange parallel rückwärts, bis der erste Unterschied $A_{i_{k'}} \neq A_{j_{\ell'}}$ auftritt.
   - Es gilt: $\ell' > k'$, also $\ell' \geq k' + 1$.
   - Dann, füge $A_{j_{\ell'}}$ nach $A_{i_{k'}}$ in I ein. Da die Elemente $A_{j_{\ell'}}$ bis A_{j_ℓ} in J zulässig sind, sind sie es auch in I.

I	A_{i_1}	A_{i_2}	$\ldots$		$A_{i_{k'}}$		$A_{i_{k'+1}}$	$\ldots$	A_{i_k}
D $\geq$	1	2	3		k'		$k'+1$	$\ldots$	k
	$\neq$	$\neq$	$\neq$		$\neq$	$\uparrow$	$=$	$=$	$=$
D $\geq$	1	2	3	$\ldots$	$\ell'-1$	ℓ'	$\ell'+1$	$\ldots$	ℓ
J	A_{j_1}	A_{j_2}	$\ldots$	$\ldots$	$A_{j_{\ell'-1}}$	$A_{j_{\ell'}}$	$A_{j_{\ell'+1}}$	$\ldots$	A_{j_ℓ}

Minimierung der Gesamtstrafe

- Für nicht rechtzeitig bearbeitete Aufträge A_i muss eine Strafe P_i bezahlt werden.
- Ziel: Minimierung der Summe aller Strafen.
- Minimale Strafe wird erreicht, wenn die Strafe für die rechtzeitig bearbeiteten Aufträge maximiert wird.
- Gesucht: Zulässige Menge von Aufträgen mit maximalem Gewicht.

Greedy-Ansatz

1. Sortiere die Aufträge in nicht-aufsteigender Reihenfolge ihrer Strafen.
2. Setze $S := \emptyset$
3. Für jeden Auftrag A in obiger Reihenfolge
4. Wenn $S \cup \{A\}$ zulässig ist
5. Setze $S := S \cup \{A\}$.

S ist eine Optimale Lösung.

Greedy-Algorithmus für Matroide

Algorithmus 3 : Greedy–Methode für ein Maximierungsproblem Π über $(M, \mathcal{U})$, $|M| = n$

Sortiere M absteigend. Die Sortierung sei $\ell_1, \ell_2, \ldots, \ell_n$.;
$I^* \leftarrow \emptyset$;
für $i = 1, \ldots, n$ **tue**
    wenn $I^* \cup \{\ell_i\} \in \mathcal{U}$ **dann**
        $I^* \leftarrow I^* \cup \{\ell_i\}$;

Schlüssel zur Optimalität der Greedy-Methode

Lemma

Sei ℓ das erste Element, das zu I^* hinzugefügt wird. Dann existiert eine optimale Lösung, die x enthält.

Beweis.

- Sei $J \in \mathcal{U}$ irgendeine optimale Lösung.
- Es gilt $\{\ell\} \in \mathcal{U}$.
- Wende „Basisergänzungssatz“ $|J| - 1$ mal auf $\{\ell\}$ an; das Ergebnis sei I^*.
- Es gilt $I^* = J \setminus \{k\} \cup \{\ell\}$.
- **Behauptung**: k hat kein größeres Gewicht als ℓ.
- **Beweis**: Sonst wäre im ersten Schritt k ausgewählt worden, denn es gilt: $\{k\} \subset J \in U \Rightarrow \{k\} \in U$. □